Vision-Based Path Following with Multi-Controller Architecture

Shlok Mehndiratta

December 22, 2025

Repository: `ugv_uav_control`
Platform: ROS 2 Jazzy, Gazebo Sim (Harmonic), Python 3.12
Document Version: v1.0

1 Problem Formulation

1.1 Original Problem Statement

The fundamental problem addressed by this project is vision-based autonomous path following for a differential-drive ground robot (TurtleBot3 Burger) using an overhead camera perspective.

The system is required to:

- Localize the robot without relying on GPS or SLAM, using fiducial marker detection.
- Extract navigable path geometry from camera imagery.
- Generate waypoints that guide the robot along the path centerline.
- Execute control commands that minimize tracking error while respecting kinematic constraints.

1.2 Constraint Hierarchy

1.3 Problem Evolution

The scope of the problem evolved through the following stages:

- **Phase 1**: Static overhead camera with direct path following.
- **Phase 2**: UAV-mounted camera for future mobile observation.
- **Phase 3**: Multi-controller architecture with modular selection.

Table 1: Constraint hierarchy governing the TurtleBot3 platform

Constraint Type	**Mathematical Form**	**Physical Interpretation**
Non-holonomic	$\dot{x}\sin\theta - \dot{y}\cos\theta = 0$	No lateral motion
Velocity bounds	$0 \le v \le 0.22$	TurtleBot3 motor limits
Angular rate bounds	$\lvert\omega\rvert \le \omega_{\max}$	Actuator limits
Wheel slip	Implicit friction model	$\mu = 1000$ assumption

2 Modeling Assumptions

2.1 Robot Kinematic Model

The TurtleBot3 Burger is modeled as a planar unicycle system with state

$$\mathbf{x} = \begin{bmatrix} x & y & \theta \end{bmatrix}^T \in \mathbb{R}^2 \times S^1,$$

where (x, y) denotes the robot position in the world frame and θ is the heading (yaw) angle.

Robot Parameters The physical parameters used in the model are:

- Wheel separation: $L = 0.160$ m
- Wheel radius: $r = 0.033$ m
- Mass: $m = 0.825$ kg
- Maximum linear acceleration: $a_{\max} = 1.0$ m/s^2

Differential-Drive to Unicycle Mapping The relationship between differential wheel velocities and unicycle control inputs is given by

$$v = \frac{r(\omega_R + \omega_L)}{2}, \qquad \omega = \frac{r(\omega_R - \omega_L)}{L},$$

where ω_R and ω_L denote the angular velocities of the right and left wheels, respectively.

2.2 Camera Model

The overhead camera is modeled using an ideal pinhole projection model. The assumed camera parameters are:

- Horizontal field of view: $\phi = 1.047$ rad (60°)
- Camera height above the ground plane: $h = 10.0$ m
- Image resolution: 1920×1080 px

World-to-Image Scaling The physical width of the observable ground region is

$$w_{\text{world}} = 2h \tan\left(\frac{\phi}{2}\right).$$

Using the image width w_{image}, the pixel-to-meter conversion factor is

$$s = \frac{w_{\text{world}}}{w_{\text{image}}} \approx 0.00601 \text{ m/px}.$$

Critical Assumption The camera optical axis is assumed to be perfectly vertical. Any deviation from this alignment introduces perspective and radial distortions, which are not corrected in the current implementation.

2.3 ArUco Marker Localization

Robot pose estimation is performed using an ArUco fiducial marker (ID 55, DICT_6X6_250). The marker is mounted 0.2 m above the robot base on a 0.5 m $\times$ 0.5 m planar plate.

- **Heading Estimation**: The robot heading is estimated from the orientation of the detected marker as
$$\theta_{\text{robot}} = \operatorname{atan2}\big(y_{\text{top}} - y_{\text{center}},\ x_{\text{top}} - x_{\text{center}}\big),$$
where $(x_{\text{top}}, y_{\text{top}})$ denotes the midpoint of the marker's top edge and $(x_{\text{center}}, y_{\text{center}})$ is the marker center.
- **Limitation**: Heading estimation accuracy degrades as the marker moves farther from the camera center due to perspective distortion. No geometric correction or compensation is applied.

3 Perception Pipeline

3.1 Lane Segmentation via Floodfill

The path extraction pipeline consists of the following stages:

- **Thresholding**: Binary intensity thresholding at a value of 100.
- **Seed Point Search**: Probing candidate seed points within a radial range of 30–70 px and an angular window of ± 0.6 rad.
- **Floodfill Propagation**: Connected-component extraction using a floodfill algorithm.
- **Distance Transformation**: Euclidean distance transform $\mathcal{D} : I \to \mathbb{R}^+$.

The path centerline is defined as the locus of points maximizing the distance transform:

$$\mathbf{p}^* = \arg\max_{\mathbf{p}\in\mathcal{R}} \mathcal{D}(\mathbf{p}),$$

where $\mathcal{R}$ denotes the set of pixels belonging to the extracted path region.

3.2 Waypoint Generation

Waypoints are generated in the robot body frame using a polar-to-Cartesian transformation. Given a target distance d and relative bearing α, the waypoint coordinates are

$$x = d\cos\alpha, \qquad y = d\sin\alpha.$$

The relative bearing is computed as

$$\alpha = -\left(\theta_{\text{target,image}} - \theta_{\text{robot}}\right).$$

Expressing the distance explicitly in metric units yields

$$x = d_{\text{meters}}\cos\alpha, \qquad y = d_{\text{meters}}\sin\alpha.$$

Coordinate Convention All waypoints are expressed in the robot body frame, with the following axis definitions:

- $+x$: Forward (aligned with the robot heading)
- $+y$: Left (perpendicular to heading, right-hand rule)
- Origin: Robot center

4 Control Strategy Evolution

4.1 Pure Pursuit Controller

Mathematical Foundation Pure Pursuit is a geometric path-following controller that computes a circular arc connecting the robot to a look-ahead waypoint.

The curvature of the arc is given by

$$\gamma = \frac{2y}{L_d^2}, \qquad L_d = \sqrt{x^2 + y^2},$$

where L_d is the Euclidean distance to the waypoint and y is the lateral offset in the robot frame.

The commanded angular velocity is

$$\omega = v \cdot \gamma.$$

Implementation In the implemented formulation, the curvature is computed as

$$\gamma = \frac{2y}{d^2},$$

and the angular velocity command is

$$\omega = v_{\text{target}} \cdot \gamma.$$

Design Rationale

- Simpler than Stanley control, as it does not require an explicit heading error term.
- More robust to noise in waypoint orientation.
- Exhibits natural corner-cutting behavior, which is beneficial for speed optimization.

Limitation Pure Pursuit does not explicitly correct heading error; as a result, the robot may approach the path with a misaligned orientation.

4.2 Stanley Controller

Mathematical Foundation Derived from Stanford's DARPA Grand Challenge entry, the Stanley controller minimizes cross-track error (CTE) and heading error simultaneously:

$$\delta = \psi_e + \arctan\left(\frac{k_{cte} \cdot e_{cte}}{1+v}\right).$$

The terms are defined as:

- ψ_e: Heading error (target heading minus robot heading).
- e_{cte}: Cross-track error (lateral deviation from the path).
- $k_{cte} = 2.0$: Cross-track error gain.
- v: Current forward velocity.

Implementation The cross-track error is computed as

$$e_{cte} = y\cos(\theta_{\text{target}}) - x\sin(\theta_{\text{target}}).$$

The steering command is then

$$\delta = \psi_e + \text{atan2}(k_{cte} \cdot e_{cte},\, 1+v),$$

and the resulting angular velocity command is

$$\omega = \frac{v \tan(\delta)}{0.2},$$

where 0.2 m is the approximate wheelbase.

Design Rationale

- Guarantees convergence to the path for bounded initial CTE.
- Exhibits natural velocity-dependent damping (higher speed implies weaker correction).
- Well-documented and empirically reliable behavior.

Limitation The Stanley controller assumes knowledge of the path tangent; in the current implementation, this tangent is approximated using waypoint orientation.

4.3 Model Predictive Control (MPC)

Choice Justification Model Predictive Control was introduced to overcome the limitations of geometric controllers:

- Enables planning over a finite prediction horizon.
- Explicitly enforces actuator constraints.
- Optimizes the trade-off between tracking accuracy and control smoothness.

4.3.1 State Augmentation

The kinematic state is augmented to include actuator dynamics:

$$\mathbf{x}_{\text{aug}} = \begin{bmatrix} x & y & \theta & v_{\text{act}} & \omega_{\text{act}} \end{bmatrix}^T.$$

Actuator Dynamics First-order actuator dynamics are modeled as

$$\dot{v}_{\text{act}} = \alpha_v (v_{\text{cmd}} - v_{\text{act}}), \qquad \dot{\omega}_{\text{act}} = \alpha_\omega (\omega_{\text{cmd}} - \omega_{\text{act}}).$$

The corresponding time constants are

$$\tau_v = 0.5 \text{ s}, \qquad \tau_\omega = 0.2 \text{ s},$$

chosen empirically.

4.3.2 Integration Method Evolution

Initial Approach (Forward Euler) The initial discretization uses Forward Euler integration:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + f(\mathbf{x}_k, \mathbf{u}_k)\, dt.$$

This approach introduces systematic bias at high angular velocities, causing curved trajectories to degenerate into straight-line chords.

Improved Approach (Exact Arc Integration) For $|\omega| \geq \epsilon$, with $\epsilon = 10^{-4}$, the exact integration of circular motion is used:

$$\begin{aligned} x_{k+1} &= x_k + \frac{v}{\omega}\big[\sin(\theta_k + \omega dt) - \sin\theta_k\big], \\ y_{k+1} &= y_k - \frac{v}{\omega}\big[\cos(\theta_k + \omega dt) - \cos\theta_k\big], \\ \theta_{k+1} &= \theta_k + \omega dt. \end{aligned}$$

For $|\omega| < \epsilon$, the straight-line approximation is applied:

$$x_{k+1} = x_k + v\cos\theta_k\, dt, \qquad y_{k+1} = y_k + v\sin\theta_k\, dt.$$

4.3.3 Cost Function

The MPC objective function is defined as

$$J = \sum_{k=1}^{N} \Big[w_d \|\mathbf{p}_k - \mathbf{p}_{\text{target}}\|^2 + w_h(\theta_k - \theta_{\text{target}})^2 + w_r \|\mathbf{u}_k - \mathbf{u}_{k-1}\|^2 \Big].$$

Table 2: MPC cost function weights

Weight	**Value**	**Purpose**
w_d	10.0	Minimize distance to target
w_h	2.0	Minimize heading error
w_r	5.0	Penalize control rate (smoothness)

The target heading is computed greedily as

$$\theta_{\text{target}} = \operatorname{atan2}(y_{\text{target}} - y,\ x_{\text{target}} - x).$$

This formulation directly points toward the target; a path-tangent-based reference would provide improved anticipatory behavior.

4.3.4 Optimization

Solver Sequential Least-Squares Quadratic Programming (SLSQP) is used for optimization.

Decision Variables

$$\mathbf{u} = [v_0, \omega_0, v_1, \omega_1, \ldots, v_{N-1}, \omega_{N-1}]^T.$$

Constraints

- Velocity bounds: $0 \leq v_k \leq 0.22$.
- Angular velocity bounds: $-2.0 \leq \omega_k \leq 2.0$.
- Rate constraints enforced implicitly via a soft penalty in the cost function.

5 Implementation Architecture

5.1 ROS 2 Node Graph

The system is organized as a modular ROS 2 computation graph integrating simulation, perception, planning, and control. The architecture follows a layered design that clearly separates responsibilities while maintaining deterministic data flow.

Simulation Layer

The simulation layer provides the physical and sensory environment:

- Ground robot exposing a velocity command interface.
- Aerial platform supplying an overhead visual perspective.
- Ground plane environment used for path visualization.

Middleware Bridge

A middleware bridge connects the simulator to the ROS 2 ecosystem:

- Translates simulator-specific messages into ROS 2-compatible formats.
- Converts velocity commands into stamped messages.
- Forwards aerial camera data to downstream perception nodes.

Perception and Planning

This layer extracts navigational intent from sensor data:

- Overhead image processing extracts path geometry.
- Waypoints are generated and published for downstream control.

Control Layer

The control layer executes motion commands based on waypoint input:

- Multiple interchangeable controllers (Stanley, Pure Pursuit, MPC).
- Consumes waypoint messages and outputs velocity commands.
- Commands are routed back to the simulator through the middleware bridge.

This layered organization enforces separation of concerns while maintaining a clear and deterministic flow of information.

5.2 Message Type Design Decision

Issue Encountered A mismatch was observed between the velocity command message type expected by the simulation bridge and the type originally published by the controllers.

Resolution All controller outputs were standardized to publish stamped velocity messages, including both a timestamp and a reference frame identifier.

Rationale The use of stamped messages provides several advantages:

- Enables future latency compensation and time-aligned control.
- Supports reliable data association for logging and debugging.
- Ensures compliance with recommended ROS 2 communication practices.

This design choice improves robustness and extensibility without increasing controller complexity.

5.3 Launch System Architecture

The system launch configuration is designed to maximize flexibility and runtime configurability. Its primary responsibilities include:

- **Dynamic Model Loading**: Simulation models are loaded at runtime, programmatically modified to substitute visual elements (e.g., track textures), and instantiated into the environment.
- **Controller Selection**: The active control strategy is selected via a runtime argument, enabling seamless switching between Stanley, Pure Pursuit, and MPC without code modification.
- **Environment Configuration**: Simulation resource paths are extended at launch time to ensure that custom models and assets are discoverable by the simulator.

This launch architecture decouples configuration from implementation, enabling rapid experimentation and reproducibility.

6 Constraint Geometry

6.1 Non-Holonomic Constraint

The mobile base is subject to a non-holonomic motion constraint, meaning it cannot generate lateral velocity. This restriction is captured by the standard unicycle kinematic model:

$$\dot{x} = v\cos\theta, \qquad \dot{y} = v\sin\theta.$$

There is no independent control over $\dot{x}$ and $\dot{y}$; translational motion is constrained to lie strictly along the instantaneous heading direction.

Consequence Path planning and control algorithms must explicitly account for a minimum achievable turning radius. Given a bounded angular velocity, the minimum turning radius is

$$r_{\min} = \frac{v}{\omega_{\max}} = \frac{0.22}{2.84} \approx 0.077 \text{ m}.$$

This constraint limits the maximum curvature of feasible trajectories and rules out instantaneous lateral motion.

6.2 Actuator Constraints

Actuation limits are imposed by the differential-drive motor model, which enforces saturation on wheel speeds. Control inputs are expressed as linear and angular velocity commands.

Command Structure The accepted command interface consists of:

- **Linear velocity**: Forward translational speed, subject to motor saturation limits.
- **Angular velocity**: Yaw rate, internally converted into differential wheel velocities.

These actuator constraints bound the feasible control inputs and must be respected by higher-level planners and controllers to ensure physically realizable motion.

7 Numerical Methods Analysis

7.1 Discretization Error

Forward Euler integration exhibits a local truncation error of order $O(dt^2)$ per integration step, which accumulates to a global error of order $O(dt)$ over a finite prediction horizon.

For a sampling period of $dt = 0.2$ s and a prediction horizon of $N = 5$, the following observations apply:

- The maximum trajectory drift over the horizon becomes significant for high-curvature paths.
- Exact arc integration eliminates this discretization error for motion segments with constant (v, ω).

7.2 Optimization Convergence

The Sequential Least-Squares Quadratic Programming (SLSQP) algorithm does not guarantee convergence for non-convex optimization problems. In this implementation, the cost function is non-convex due to:

- Trigonometric terms appearing in the state propagation equations.
- Angle wrapping effects in the heading error formulation.

Mitigation Strategy Convergence robustness is improved by initializing the optimizer using the solution from the previous control step. Additionally, the maximum number of solver iterations is capped at 20 to bound computation time.

7.3 Numerical Stability

The exact arc integration formulation contains the term $\frac{v}{\omega}$, which becomes ill-conditioned as $\omega \to 0$.

Stability Handling To maintain numerical stability, the integration scheme switches to a straight-line approximation when

$$|\omega| < 10^{-4}.$$

This conditional handling prevents numerical singularities while preserving accuracy in the low-curvature limit.

8 Limitations and Failure Analysis

8.1 Perception Limitations

Table 3: Limitations of the perception pipeline

Limitation	**Manifestation**	**Mitigation**
Binary thresholding	Fails under variable lighting conditions	Adaptive thresholding (not implemented)
Single seed point	May select an incorrect lane at intersections	Multi-seed strategy (not implemented)
No marker tracking	Re-detection required at every frame	Extended Kalman Filter (EKF) state estimation (not implemented)

These limitations indicate that the current perception pipeline is primarily reactive and lacks temporal consistency.

8.2 Control Limitations

Table 4: Limitations of the control subsystem

Limitation	**Manifestation**	**Root Cause**
Single waypoint horizon	Poor anticipation of upcoming path curvature	Perception module publishes only one waypoint
No state estimation	Assumes ideal command execution	Odometry feedback not integrated
Soft rate constraints	Potential for jerky motion	Hard constraints computationally expensive in Python

These control limitations reduce tracking smoothness and robustness, particularly in high-curvature or dynamically changing scenarios.

8.3 System-Level Limitations

Beyond perception and control, several system-level constraints limit overall capability:

- **Latency**: Python-based MPC optimization may exceed 50 ms in worst-case scenarios.
- **Static UAV assumption**: The system assumes a hovering aerial platform, with no coordinated aerial–ground motion.
- **No obstacle avoidance**: Path-following logic assumes a clear environment without dynamic or static obstacles.

9 Future Research Directions

9.1 Immediate Extensions (Engineering)

Several near-term engineering extensions can significantly improve system robustness, performance, and deployability.

- **Warm-start optimization**: Initialize the optimizer using a shifted version of the previous solution to improve convergence speed and reduce computational overhead.
- **Path spline fitting**: Fit a cubic spline to the extracted path centerline in order to ensure trajectory continuity and smooth curvature profiles.
- **Odometry integration**: Fuse wheel odometry with visual marker-based localization to obtain more robust and drift-resistant state estimation.

9.2 Theoretical Directions

Beyond implementation improvements, several control-theoretic extensions can provide stronger analytical guarantees and cleaner problem formulations.

- **Frenet-frame MPC**: Reformulate the optimization problem in path-relative (Frenet) coordinates, enabling more interpretable cost terms and explicit separation of longitudinal and lateral errors.
- **Terminal cost**: Introduce a terminal penalty $\phi(\mathbf{x}_N)$ to improve closed-loop stability and convergence guarantees.
- **Explicit MPC**: Pre-compute the control law offline to enable deterministic real-time execution with bounded computational complexity.

9.3 Learning-Based Extensions

Learning-based approaches offer the potential to augment or partially replace model-based components in complex environments.

- **Cost function learning**: Apply inverse reinforcement learning to infer MPC cost weights from expert demonstrations.
- **Path extraction**: Employ semantic segmentation networks to enable robust path perception in visually complex or unstructured environments.
- **End-to-end control**: Use imitation learning to directly map sensory observations to control commands, bypassing explicit path extraction and geometric modeling.

9.4 Multi-Agent Coordination

Extending the framework to cooperative multi-agent scenarios opens several additional research directions.

- **UAV–UGV cooperative planning**: Leverage aerial sensing to provide long-horizon look-ahead information for ground vehicle navigation.
- **Dynamic path modification**: Allow the aerial agent to modify visual cues or markers in real time to influence ground navigation behavior.
- **Distributed MPC**: Perform coordinated trajectory optimization across multiple agents while respecting individual dynamics and communication constraints.

Appendices

A Symbol Definitions

Symbol	**Definition**	**Units**
x, y	World-frame position	m
θ	Heading (yaw angle)	rad
v	Linear velocity	m/s
ω	Angular velocity	rad/s
δ	Steering angle (Stanley controller output)	rad
γ	Path curvature	1/m
e_{cte}	Cross-track error	m
ψ_e	Heading error	rad
L	Wheel separation	m
r	Wheel radius	m
dt	Discretization time step	s
N	MPC prediction horizon	–
w_d, w_h, w_r	Cost function weights	–

A Sensor Specifications

Sensor	**Type**	**Topic**	**Rate**
Overhead Camera	RGB	/overhead_camera/image	30 Hz
IMU	6-axis	/imu	200 Hz
LiDAR	2D	/scan	5 Hz
Wheel Encoders	Quadrature	/joint_states	–